Dynamic analysis of structures on multicore computers

Mackie, R.I.

Published in:
Advances in Engineering Software

DOI:
10.1016/j.advengsoft.2013.03.006

Publication date:
2013

Document Version
Peer reviewed version

Advances in Engineering Software xxx (2013) xxx–xxx

Contents lists available at SciVerse ScienceDirect

Advances in Engineering Software

journal homepage: www.elsevier.com/locate/advengsoft

Dynamic analysis of structures on multicore computers – Achieving efficiency through object oriented design

R.I. Mackie *

Civil Engineering, School of Engineering, Physics and Mathematics, University of Dundee, Dundee DD1 4HN, UK

ARTICLE INFO

Article history:
Available online xxxx

Keywords:
Component-oriented
Object-oriented
Eigenproblems
Transient analysis
Seismic analysis
Parallel computing
Distributed computing

ABSTRACT

The paper examines software design aspects of implementing parallel and distributed computing for transient structural problems. Overall design is achieved using object and component oriented methods. The ideas are implemented using .NET and the Task Parallel Library (TPL). Parallelisation and distribution is applied both to single problems, and to solving multiple problems. The use of object-oriented design means that the solvers and data are packaged together, and this helps facilitate distributed and parallel solution. Factory objects are used to provide the solvers, and interfaces are used to represent both the factory objects and solvers.

1. Introduction

There have been many changes in computing hardware and software. One of the most recent is that virtually all computers are now multi-core, typically dual or quad core. This has implications for the design of software, which has yet to be fully realised. Techniques for parallel computing have largely been developed for high-performance computing (HPC) on super computers or clusters of workstations. This has some relevance to the new world on everyday computers, but there are other possibilities as well. One of these is the user-interaction and software. HPC is largely geared at solving highly complex problems that require massive computing resources and take a long time. This has some relevance to normal computing as desktop computers are now capable of solving much more complex problems than used to be the case. Furthermore computers are linked together on intranets and the internet, so the creation of clusters of computers is relatively easy. However, the usability of engineering software, and the way the software can be used in design is equally important, and the power and architecture of current computers changes what is now possible. This paper will look at some of the possibilities in the area of dynamic analysis of structures, with particular focus on seismic engineering. However, the ideas presented are much more widely applicable. The paper will also emphasise key software design decisions that facilitate the exploitation of the modern computing environment.

Current software developments are addressing the new environment, in particular version 4.0 of Microsoft's .NET framework. The .NET framework was developed with distributed and multi-threading computing in mind, and has had facilities for simplifying software development for this world. Version 4.0 has introduced the Task Parallel Library (TPL) [1] to facilitate software for multi-core computing. There is a tendency to think that HPC should be performed using MPI and on Linux machines. MPI and Linux definitely have their place, but the reasons for using .NET in the current work are:

- Windows is the most commonly used operating system. Now on supercomputers Unix/Linux based operating systems are by far the most common. However, the overarching motivation behind the current work is that parallel and distributed computing are now part of the mainstream computing world. Therefore it is appropriate to consider the application of technologies designed for mainstream computing.
- The .NET infrastructure is available on all Windows computers, so there is no need to install any further software.
- .NET provides parallelisation and distribution in an object and component oriented fashion, so it is consistent with the overall design philosophy.

Dynamic problems in structures are among the more expensive in computational terms, they also generate an incredible amount of data. So there are two problems that need to be addressed in software development: efficient numerical problems, and data handling. Work on parallelism typically focuses on parallelising

* Tel.: +44 1382 384702; fax: +44 1382 384816.
E-mail address: r.i.mackie@dundee.ac.uk

0965-9978/$ - see front matter © 2013 Published by Elsevier Ltd.
http://dx.doi.org/10.1016/j.advengsoft.2013.03.006

the solution of a single problem. However, it can also sometimes be useful to solve several problems at once, and in terms of parallelism this is actually simpler as one can expect to achieve greater speed-up. The work described in this paper will apply parallelisation to both aspects. While earthquake engineering provided the motivation for this work, the software engineering aspects of this work are much more widely applicable.

The motivation for the current work is the analysis of space structures under seismic loading. On the one hand, the object-oriented implementation of modal and transient analysis algorithms will be examined. Earthquakes are by their very nature uncertain, so it is useful to look at the behaviour of a structure under several different earthquakes. The new environment makes this much more feasible, and the design of software to facilitate this will be described.

The advantages of parallel and distributed processing can be used in various ways:

1. Application to individual algorithms.
2. Application to solving multiple problems simultaneously.
3. Overall program design.

This paper will look at all three of these aspects, making use of object and component oriented programming design. It should be noted that the primary emphasis of the current paper is on program design rather than numerical efficiency. Therefore the focus is on explaining design approaches to make implementation of flexible parallel and distributed programs easier on mainstream computers.

2. Literature review

Modal analysis and time stepping are both well established, and there are many algorithms for solving these problems. A good description of general techniques relevant for finite element analysis can be found in Bathe [2]. There has been a significant amount of work on parallelisation of both eigenproblems solvers and time stepping algorithms. The main methods used for eigensolution methods are sub-space iteration, the Lanczos method, and component mode synthesis.

PARPACK [3] is a parallel package for large eigenvalue problems. Wu and Simon [4] implemented a parallel Lanczos method for the symmetric generalised eigenvalue problem, and used MPI. Guarracino et al. [5] used a block Lanczos algorithm to solve eigenproblems on multiple computers. Honglin et al. [6] have used a parallel implementation of the sub-space iteration method, and applied it to non-linear problems.

Cross [7] and Aoyama and Yagawa [8] both used parallel implementations of the component mode synthesis method. They reported near ideal speed-up on massively parallel computers. However, the work was based on one dimensional splitting of the structure into sub-domains.

Li et al. [9] described the uses of supercomputers and Nastran and Patran, and use IRAM (implicit restarted Arnoldi method) for symmetric eigenproblems, and achieved up to 75% speed-up efficiency.

Most of the work has used MPI or other parallel methods. There is very little on the object or component oriented implementation, the work of Heng and Mackie [10] being an exception to this. A more general consideration of the use of MPI, Java and C# in parallel and distributed computing can be found in Mackie [11]. It should be noted that another area that is receiving considerable attention is the use of Graphics Processing Units (GPUs) for general purpose computing, seeking to take advantage of the fact that GPUs have many cores and are high-performance [12].

Manolis et al. [13] uses sensitivity and stochastic modelling to help develop retrofit strategies for structures under seismic loading. Such work requires many analyses. Dere and Sotelino [14] also commented on the need for multiple analyses for establishing response spectra in non-linear problems. They implemented a parallel sub-domain solution approach using a group-implicit algorithm. Fu [15] also used a sub-domain approach, but with an overlapping domain algorithm,

3. Object oriented program design

The work in this paper will be described within the context of seismic engineering and space structures, but the work described herein is not limited to this problem area. Rather, it is used as a vehicle for demonstrating various program design principles and methods.

As noted in the introduction, there are various ways in which programs can take advantage of parallelism and distributed processing. The most obvious, and probably the most common, is the application of parallelism to individual problems. If multiple problems need to be solved, then these too can be done in parallel. Within the context of seismic engineering, earthquakes are by their very nature stochastic, and the precise earthquake a structure may have to endure is not known. So it can be useful to subject a structure to a variety of earthquakes. In addition, there is the general design of the program. The presence of multiple processors means that the program can do several tasks at once, so often it is not necessary for the program to stop completely while doing some tasks. This can help with the overall usability and flow of the program. The software described in thus paper was written using C# and the .NET environment, version 4. C# is object and component oriented, and version 4 of .NET has introduced the Task Parallel Library (TPL).

3.1. Parallelising of individual algorithms

The algorithms involved in seismic analysis are: (i) determination of the modal frequencies and mode shapes; (ii) modal superposition; (iii) transient analysis.

Determination of the vibration modes by the subspace iteration method was examined in [16]. The algorithm itself can be parallelised. Further parallelisation can be achieved by using domain decomposition. The paper described the use of a design pattern which has also been used in the design of software for iterative solvers [17]. The work described in [18] has been modified to use the TPL, but the overall design remains the same. [10] described the implementation of component mode synthesis. Work by others has parallelised the Lanczos algorithm.

The results of the modal analysis can be used in the mode superposition method, though naturally this applies only to linear problems.

Transient analysis can be applied to linear and non-linear problems. Since the focus of the current work is on software design aspects, the work in the current paper is limited to the linear case. The Hilber–Hughes–Taylor algorithm is used, and the design pattern used for eigensolution and iterative solvers [16,17] is adopted.

The key feature of the design pattern is the separation of the algorithm from the data. The algorithm for a particular problem remains the same, but it may be implemented for many different data structures. For instance the standard solution would be the use of a single domain, but the algorithm can also be implemented for the domain decomposition case. Furthermore, the data may be stored locally or remotely, or a mixture of the two. Even then, for each of these cases there are a multitude of sparse data storage schemes that can be used. However, despite all these variations,

Please cite this article in press as: Mackie RI. Dynamic analysis of structures on multicore computers – Achieving efficiency through object oriented design. Adv Eng Softw (2013), http://dx.doi.org/10.1016/j.advengsoft.2013.03.006

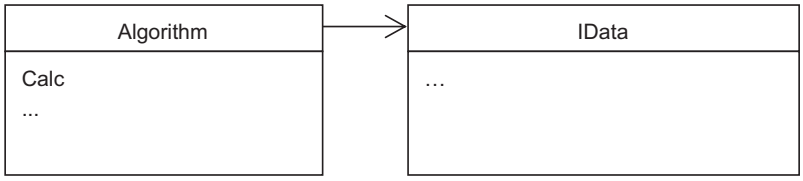

Fig. 1. Algorithm data structure.

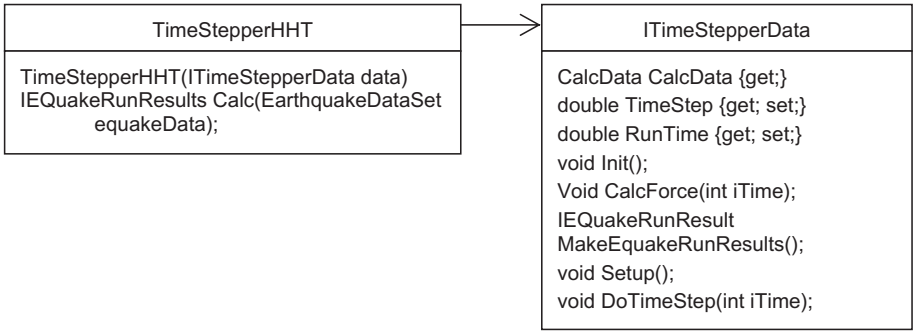

Fig. 2. Class structure for `TimeStepperHHT`.

the basic algorithm itself remains the same. Using this design pattern means that the algorithm is isolated from the effects of changes in data structure. This is an example of interfaces handling complexity, by isolation.

Fig. 1 shows the general class structure, and Fig. 2 the class structure for this particular case. Fig. 1 could be viewed as a design pattern, the key feature is that the algorithm and data are separated. `ITimeStepperData` is an interface that represents the finite element model, and data associated with the algorithm. An interface contains methods only with no implementation, it defines what something can do without giving any implementation. So it includes things like the stiffness and mass matrices. Two versions of the interface have been created, one that just uses a single matrix for the stiffness matrix and one for the mass matrix; and one that uses the domain decomposition approach. The latter approach allows parallelism to be used. The objects can be local or distributed on remote computers.

`TimeStepperHt` has the following definition

```
public class TimeStepperHHT: MarshalByRefObject,
  ITimeStepper,
  IDisposable
{
  ITimeStepperData data;
  IEQuakeRunResults runResults;
  public TimeStepperHHT(ITimeStepperData _data)
  {
    data = _data;
  }
  public void Calc(EarthquakeDataSet
  earthquakeData)
  {
    ...
  }
...
}
```

The object is initialised with an `ITimeStepperData` object. It does not know whether this is local or remote, or whether it is using a single domain or domain-decomposition. All it knows is that it implements the `ITimeStepperData` interface, and that is all that it needs to know. `TimeStepperHHT` implements the `ITimeStepper` interface:

```
public interface ITimeStepper
{
  void Calc(EarthquakeDataSet earthquakeData);
  IEQuakeRunResults RunResults { get; }
}
```

This is a very simple interface, essentially saying that calculations will be carried out on a data set, and that the results of the calculations can be obtained. This means that it does not specify the time stepping algorithm used. `TimeStepperHHT` uses the HHT algorithm, and the `Calc` method itself has the following code:

```
public void Calc(EarthquakeDataSet
  earthquakeData)
{
  ...
  runResults = data.MakeE
  QuakeRunResults(noTimeSteps,
  earthquakeData);
  data.Init();
  data.Setup();
  for (int i = 1; i < noTimeSteps; i++)
  {
    data.DoTimeStep(i);
  }
  runResults.CalcForce();
}
```

data is an `ITimeStepperData` object. First it creates a results object, this also gives the data object information on the earhtquake. Then it initialises the data object. The standard version of `ITimeStepperData` creates the stiffness and mass matrices in this method. The domain decomposition version does the same sort of thing, but for each sub-domain. Setup then prepares the object for doing the time-stepping. Next the main time-stepping loop occurs. Finally

Please cite this article in press as: Mackie RI. Dynamic analysis of structures on multicore computers – Achieving efficiency through object oriented design. Adv Eng Softw (2013), http://dx.doi.org/10.1016/j.advengsoft.2013.03.006

the `CalcForce` method calculates the forces in all the members of the space truss.

This same design pattern was first used in the context of iterative solvers [16]. Many different data structures, and variations of the algorithm (different pre-conditioning strategies, Schur complement approaches, etc.) have been used, and the basic algorithm object has remained unchanged. All the variations were handled by developing new data objects, thus demonstrating the effectiveness of the approach in isolating areas of complexity.

3.2. Solving multiple problems

As well as solving individual problems, parallelism and distributed computing can also be used to solve several problems in parallel. One situation where this can be useful is in subjecting a structure to several earthquakes. Earthquakes by their very nature are unpredictable, so it can be beneficial to subject the structure to a series of earthquakes, either based on previously recorded earthquakes, or artificially generated accelerograms.

There are two aspects to the software design, parallelism and distributed processing. Parallelism is implemented using the .NET Task Parallel Library (TPL). At a basic level this has some similarities to OpenMP, in the way that it allows easy parallelisation of for loops. However, there are many more features to facilitate program control, such as cancellation tokens. This latter aspect will be discussed further in the next section.

The overall scheme is shown in Fig. 3. The client obtains the earthquake data sets from a database. It then obtains a factory object, called `solverFactory`. Factory objects are commonplace in object-oriented programming and are a well-established design pattern. `solverFactory` implements a very simple interface `IDynamicSolverFactory` that says it can return an object of type `ITimeStepper`. Since a factory object is used the code is independent of the type of solver used, and whether it is local or remote. Then for each data set it executes `DoTimeStepping`. The code used for solving several problems is as follows:

```
MyLinkedList<EarthquakeDataSet> dataSets;
Parallel.ForEach(dataSets, DoTimeStepping);
```

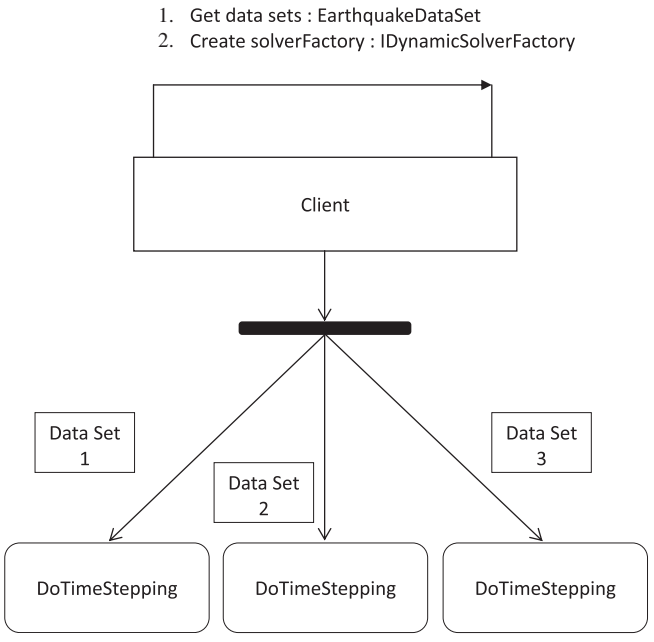

Fig. 3. Solving several problems.

`EarthquakeDataSet` is a class that contains the data for an earthquake, in particular the accelerograms. `dataSets` is the linked list of several earthquakes that the structure is to be subjected to. `Parallel.ForEach` is a parallelised version of the `ForEach` keyword. The basic `ForEach` simply iterates sequentially through the items in list, carrying out the required operations in serial mode. The parallel version iterates through the list, but does this in parallel. A new thread is used for each item in the list, so the operations are performed in parallel. .NET has a threadpool, and takes threads from this pool. This means new threads are not created, but are already available, so the process is efficient.

For each item in the list, data, it executes `DoTimeStepping(EarthquakeDataSet data)`, and does this in parallel. The parallel loop waits for the operations for each data set to complete before exiting from the loop. A similar construct exists for `for` loops, TPL also allows them to be used with greater control as well.

This simple programming construct works either on a single machine with multiple cores, or for distributed operation.

The main part of the code for `DoTimeStepping` is

```
void DoTimeStepping(EarthquakeDataSet data)
{
  ...
  TimeStepperHHT timeStepper =
    solverFactory
  .GetTimeStepper(trussModel,calcData);
  IEquakeRunResults
  results = timeStepper.Calc(data);
  results.Set(trussModel, data);
  lock(resultsLock)
  {
    resultsSets.Add(results);
  }
}
```

There are a number of points to pay attention to. The first is that a factory object, called `solverFactory` in the above code, is used to supply the solver, called `timeStepper`. This means that the code here works whether the solver is local or remote. The code also works for different time stepping algorithms. All the code is concerned with is that `solverFactory` returns an object that will carry out the time-stepping operations. It is not concerned with how it carries them out. The use of factory objects is a standard design pattern in object-oriented programming. In the current context its main advantage is that the details of creating and maintaining remote solvers is all contained within the factory object, and so the rest of the program is isolated from these details. In particular, for the distributed case the factory object has a list of available host computers and distributes the work around these hosts. It also sponsors the remote objects to ensure that they remain "live". Moreover, the factory could take implement load balancing.

If all the calculations were to be performed on the same computer, then the factory returns a standard local time stepper object. If distributed solution was being used, then the factory object would create the object on a remote computer, and the calculations would be carried out on that computer. More details will be given shortly. The factory object could return either a solver using standard solution, or one using domain decomposition.

The solver returned could use either serial or parallel solution. Which would be the most efficient would vary from case to case. For instance suppose four quad core computers were available. Then if there were four earthquakes to be analysed it might be more efficient to use a domain decomposition (i.e. parallel) based algorithm for the solution. However, if there were sixteen

Please cite this article in press as: Mackie RI. Dynamic analysis of structures on multicore computers – Achieving efficiency through object oriented design. Adv Eng Softw (2013), http://dx.doi.org/10.1016/j.advengsoft.2013.03.006

earthquakes then using a serial solver and relying purely on the coarse-grained parallelism could be better.

A second point is that the solver is encapsulated in an object, rather than simply coded in a method. This has several advantages. One is that there is a different solver object for each earthquake data set, and so data integrity is handled easily. A second is that if a domain decomposition solver is used, then if this exists on a remote computer, the remote computer automatically uses parallelism when executed on the remote machine. So we have an excellent example here of one of the advantages of the object-oriented approach, namely that it packages methods and data together into a single entity, an object, and this makes for better program design, and helps facilitate distributed implementation. The reason it does this is that it helps with data integrity. Good high level design decisions makes low level details easier to handle.

`IEquakeRunResults` is an interface that defines the results set returned. The details would be different for standard and domain decomposition models. In the current program, if a remote computer was used, the results are returned to the client. However, since interfaces are used, it would be perfectly possible for the results to remain on the remote computer, possibly stored on a remote database. So again we see the value of interfaces in program design. All that the client needs to know is that results are somehow or other accessible.

The results are then added to `resultsSets`. The lock keyword is used here to ensure that only one results set is added to `resultsSets` at a time, and so data conflicts are avoided. This is a simple illustration of a fundamental fact of parallel computing. There are various mechanisms and constructs, such as those available in TPL, that make parallel programming easier. However, the logic of parallel programming is inherently more complex than that of serial programming, and the programmer has to understand what is going on. Currently at least, there is a limit to how much programming frameworks can help.

Distributed processing is implemented using remote objects. Details on some aspects of remote objects have been given elsewhere [18], and recently on deployment [19]. This latter reference described various ways in which remote objects can be implemented. The approach used here is for the solver programs to be on the remote computers, and the client program to access them via an interface. When it starts the host program registers itself on a central register program, as shown in Fig. 4.

This program just keeps a record of which hosts are available. The program then just sits there until a client seeks to obtain a factory object from it. The use of interfaces creates a logical (and physical in the case of remote objects) wall between the client program and the solver objects.

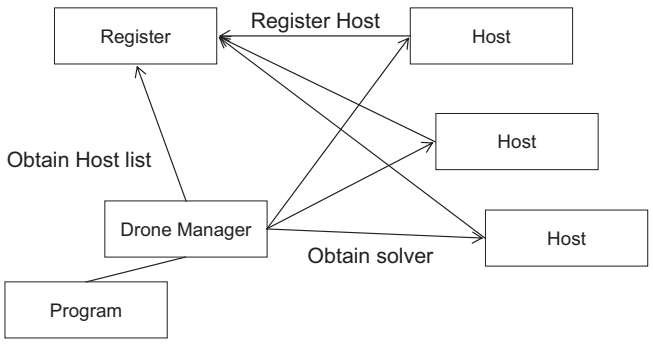

Fig. 4. Distributed host architecture.

When the client program wants to create an object on the host it uses an object called `DroneManager`. The client then activates the factory object, which will exist on the remote computer, and the factory object is then used to create the solver object. This solver object will also exist on the remote computer, and the calculations are carried out on that computer. It would be possible for more sophisticated load balancing to be placed in DroneManager. However, currently it just goes round the list of available hosts in turn. Data transfer costs are an issue in distributed systems and results of tests on clusters of Windows computers are reported elsewhere [11,19]. Further data when using WCF (Windows Communication Foundation) has also been published by the author [20].

A factory object is used because an interface is used to represent it, and the object can only be created using the default constructor. So members of the factory object then enable the solver objects to be created with parameters.

3.3. Overall program design

While the preceding sections describe the main aspects of using parallelism and distributed processing, there are other aspects to program design that help with the overall operation of the program. The traditional mode of operation of programs is that they do one operation, even though that operation itself may use parallelism, at a time; this is a throwback to the time when most computers were predominantly serial. Programs can be much more responsive and flexible if tasks are carried out in separate threads, so that no task completely freezes the program. This does introduce more complexity as there needs to be proper co-ordination between tasks and program control. An earlier paper [21] described some mechanisms for handling this scenario. .NET has a number of features which enable these aspects to be handled properly.

A simple example of the difference between the serial and event-driven approach can be seen by considering the current program. Within the program if modal super-position is to be used, then there are various tasks that can be carried out. These include:

1. Read the model data file.
2. Calculate vibration modes of the structure.
3. Read in the earthquake data files.
4. Carry out transient analysis.
5. Visualise the results.

The tasks can be carried out in a serial manner, with each task carried out one after another. However, there is no logical necessity for things to be done this way. The earthquake data and model data can be read independently of each other, and calculation of the vibration modes is independent of the earthquake data.

In order to take advantage of the ability to do tasks in a non-serial manner various things need to be done. First the tasks need to be executed in different threads. Secondly, the tasks may need to feed back progress to the main program. Thirdly the main program may need to cancel the tasks. .NET provides features which help facilitate this.

One of these is the `BackgroundWorker` class. This is a class which manages some of the aspects of carrying out calculations in separate threads, and it uses various events as the means of achieving this. The most important is that a method is defined which actually does the work (e.g. calculate the vibration modes). Of most relevance in the current section is that it has two other events, one related to progress and one related to completion. So the worker method can fire the `BackgroundWorker ReportProgress` event at various junctures and the main program can decided how to report, and possibly respond to, the progress.

Please cite this article in press as: Mackie RI. Dynamic analysis of structures on multicore computers – Achieving efficiency through object oriented design. Adv Eng Softw (2013), http://dx.doi.org/10.1016/j.advengsoft.2013.03.006

Similarly `BackgroundWorker` method fires a completion event when it has finished its calculations and the main program can then take appropriate action.

The TPL has `CancellationTokens`. These can be passed around various objects, so if some part of the program decides certain tasks need to stop, it can set the appropriate token, and any tasks that have possession of that token, will take suitable action the next time they check the state of the token. Note that the task is not forced to stop, rather the program tells it that it would like it to stop. The reason for this is that only the task knows how to stop gracefully.

The key features of these control and communication mechanisms are subscribing to events, firing events, and communication tokens. As described briefly, .NET has a number on inbuilt facilities for implementing these mechanisms. However, the same design approach can and is used in other environments.

4. Results

The subspace iteration method was used on two square-on-square offset space trusses. One is 20×20 m, and the other 60×60 m. Fig. 5 shows a picture of the 20×20 space truss. The truss has pin supports at each of the four corners.

The 20×20 space truss model had 2048 bar elements, 545 nodes and 1623 degrees of freedom. The 60×60 space truss had 12,800 elements, 3281 nodes and 9831 degrees of freedom. Both structures were subjected to seismic excitation, and transient analysis was used to predict the response. In the current work the analysis was limited to the linear case as the primary focus was on the software engineering aspects. The calculations were carried out on a variety of machines. These were:

A – One Intel Core i7, 2.80 GHz quad core, 6 GB Ram, Windows 7 (64 bit).
B – Intel Core 2 Quad Q9450, 2.66 GHz, 3 GB Ram Windows 7 (32 bit).
C – Cluster of Intel duo E8400 3 GHz, 3 GB Ram, Windows XP.

Various combinations of machines and calculation runs were executed. Fig. 6 shows the speed up gained by using the domain decomposition version of the time stepper, these were carried out on Machine A. As can be seen, a speed up of approximately two is achieved. The structure was split into four sub-domains.

Fig. 7 shows the speed-up achieved solving multiple problems. 4(A) and 8(A) involved solving 4 and 8 problems, respectively, on machine A. 8(A + B) used machines A and B for 8 problems. As there are only 4 physical cores on both machines it is not surprising that there is little change in speed-up in going from 4(A) to 8(A). There is some further speed-up for 8(A + B), it should be noted that machine B is a little slower than machine A. The speed up was calculated by comparing the time for 4 (or 8 problems) with 4 (or 8) times the time for 1 problem.

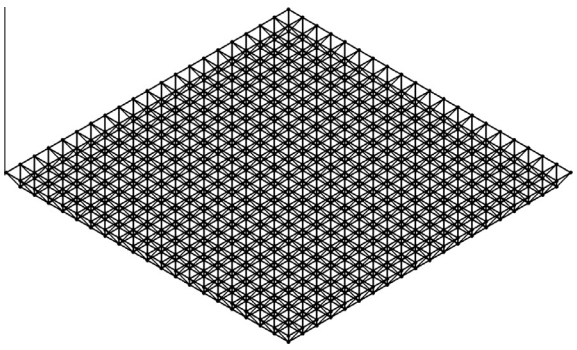

Fig. 5. 20×20 Space truss.

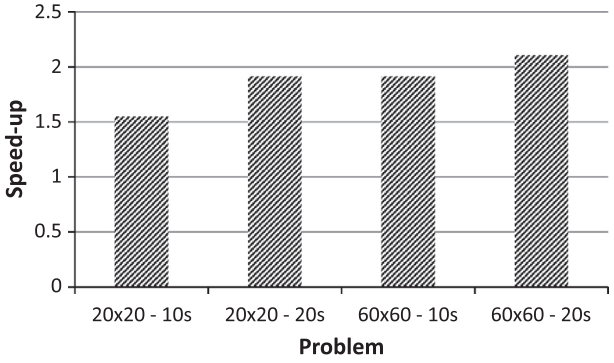

Fig. 6. Speed-up achieved using domain decomposition – Machine A.

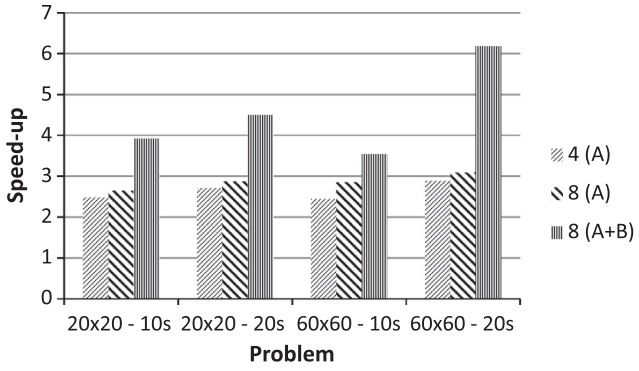

Fig. 7. Speed-up for multiple problems.

For multiple problems the serial time stepping solver was used, rather than the domain decomposition method. The reason for this was that mixing domain decomposition with solving multiple problems on a single computer was detrimental to efficiency when the number of problems multiplied by the number of domains exceeded the number of cores available. If just two problems were solved, then using A and B and domain decomposition did result in significant speed-up, with a factor of 1.44 over using domain decomposition on a single machine, 2.23 over using serial solution on a single machine.

The final batch of tests was involved in using the cluster of machines defined in C. Fig. 8 shows the speed up for the 20×20 problem, run for ten seconds. For one machine the speed-up was

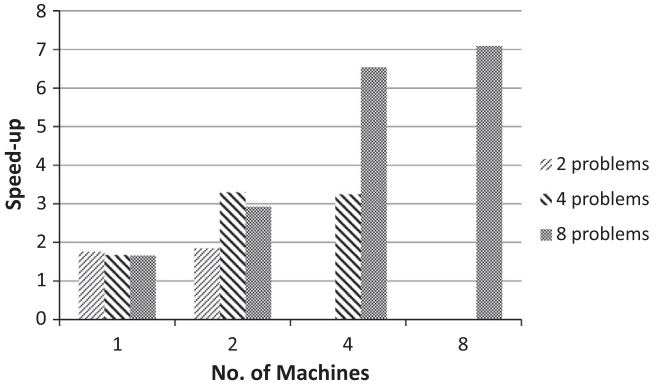

Fig. 8. Speed-up for Cluster C – 20×20 problem.

Please cite this article in press as: Mackie RI. Dynamic analysis of structures on multicore computers – Achieving efficiency through object oriented design. Adv Eng Softw (2013), http://dx.doi.org/10.1016/j.advengsoft.2013.03.006

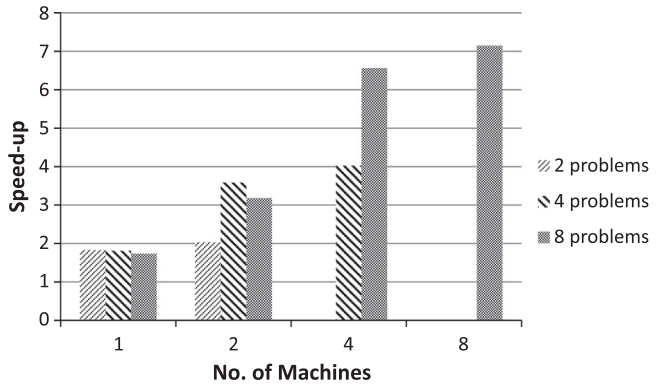

Fig. 9. Speed-up for Cluster C – 60 × 60 problem.

around 1.7 whether there were 2, 4 or 8 problems. The reason why there is any speed-up at all is that the machine is dual core. For two machines the speed-up gets close to 3 for 4 or 8 problems. With 4 machines the speed-up for 8 problems is 6.5, and this increases further to just over 7 for 8 machines. Even with 4 machines there are still 8 processors, so in principal there should be no further speed-up going from 4 to 8 machines for 8 problems. However, it does appear that using two machines is slightly better than one. The extra cache memory available may be a factor here.

Fig. 9 shows the equivalent results for the 60 × 60 problem. The pattern is the same, though with slightly higher speed-ups. In all these cases the underlying problem was linear, and the next stage will be to extend the software to solve non-linear problems.

5. Conclusions

The widespread availability of multi-core and distributed computers opens up new possibilities for general engineering software. In order to realise these advantages requires changes in the way the programs are designed.

The current paper has focused on transient analysis. The new architectures can be used to solver individual problems using parallel methods. They are particularly relevant for scenarios where multiple problems need to be solved.

Object and component oriented software design methods can be used to help facilitate the development of code that take advantage of the current hardware architectures. In particular, the use of objects helps to package the solver, so it can then be distributed to remote computers as desired.

The use of interfaces isolates areas of complexity, and means that much of the software is oblivious to the type of solver that is being used, or whether remote or local solution is being used.

Further to the direct problem of parallelism and distributed computing, there are other impacts on overall software design, and features within a framework, such as event-driven driven, can be used to enable overall control of the program, and to software enhance flexibility.

The ideas were implemented in a software program to solve seismic problems. As mentioned in the introduction, the emphasis in this paper has been on design approaches, and the algorithms and code have not been highly optimised. If better algorithms were implemented these could be "plugged" into the current framework. Even so, the speed-up obtained is reasonable.

References

[1] Freeman A. Pro. NET 4.0 parallel programming in C#. APress; 2010.
[2] Bathe K-J. Finite element procedures in engineering analysis. Englewood Cliffs, New Jersey, USA: Prentice-Hall; 1995.
[3] PARPACK. <http://www.caam.rice.edu/~kristyn/parpack_home.html>.
[4] Wu K, Simon HD. A parallel Lanczos method for symmetric generalized eigenvalue problems. Comput Vis Sci 1999;2:37–46.
[5] Guarracino MR, Perla F, Zanetti P. A parallel block Lanczos algorithm and its implementation for the evaluation of some eigenvalues of large sparse symmetric matrices on multiple computers. Int J Appl Math Comput Sci 2006;16:241–9.
[6] Honglin L, Xicheng W, Xinli L, Hailei Z, Kun Y. A parallel subspace iteration method with mdal transfer for electronic-structure calculation. In: Int conf on parallel algorithms and computing environments. The Chinese University of Hong Kong; October 8–11, 2003.
[7] Cross J-M. Component mode synthesis method and parallel computing. C R l'Academie Sci, Ser IIb: Mecanique Phys Chim Astron 1999;327:13–8.
[8] Aoyama Y, Yagawa G. Large-scale eigenvalue analysis of structures using component mode synthesis. JSME Int J Ser A 2001;44:631–40.
[9] Li L, Jin X, Li Y, Wei J. A parallel solver for structural modal analysis based on commercial FEA code. Int J Adv Manuf Technol 2005;25:199–204.
[10] Heng BCP, Mackie RI. Parallel modal analysis with concurrent distributed objects. Computers and Structures 2010;88:1444–1458. http://dx.doi.org/10.1016/j.compstruc.2008.06.002.
[11] Mackie RI. High performance computing on low cost computers: a review of parallel and distributed computing methodologies for finite element analysis. In: Topping BHV, Adam JM, Pallarés FJ, Bru R, Romero ML, editors. Developments and applications in engineering computational technology. Stirlingshire, UK: Saxe-Coburg Publications; 2010 [chapter 12].
[12] GPGPU. <http://gpgpu.org/about>.
[13] Manolis GD, Panigiotopoulos CG, Paraskevopoulos EA, Karaoulanis FE, Vadaloukas GN, Papachristidis AG. Retrofit strategy issues for structures under earthquake loading using sensitivity-optimization procedures. Earthquakes and Structures 2010;1:109–27.
[14] Dere Y, Sotelino ED. Solution of transient nonlinear structural dynamics problems using the modified iterative group-implicit algorithm. In: Topping BHV, Bittnar Z, editors. Proceedings of the third international conference on engineering computational technology. United Kingdom: Civil-Comp Press, Stirling; 2002 [paper 34].
[15] Fu C. A parallel algorithm for nonlinear dynamic finite element analysis. In: 1st Int conf on information science and engineering, Nanjing, PR China; 2009.
[16] Mackie RI. Implementing modal analysis software on multi-core computers: with application to seismic analysis of space trusses. In: Topping BHV, Adam JM, Pallarés FJ, Bru R, Romero ML, editors. Proceedings of the 10th international conference on engineering computational technology. Stirlingshire, United Kingdom: Civil-Comp Press; 2010 [paper 332].
[17] Mackie RI. Object oriented programming of distributed iterative equation solvers. Comput Struct 2008;86:511–9.
[18] Mackie RI. Programming distributed finite element analysis: an object oriented approach. Saxe-Coburg, Stirling; 2007. ISBN:978-1-874672-31-9.
[19] Mackie RI. Design and deployment of distributed numerical applications using .NET and component oriented programming. Adv Eng Softw 2009;40:665–74.
[20] Mackie RI. Application of service oriented architecture to finite element analysis. Adv Eng Softw 2012;52:72–80.
[21] Mackie RI. Using objects to handle calculation control in finite element modelling. Comput Struct 2002;80:2001–9.

Please cite this article in press as: Mackie RI. Dynamic analysis of structures on multicore computers – Achieving efficiency through object oriented design. Adv Eng Softw (2013), http://dx.doi.org/10.1016/j.advengsoft.2013.03.006